ASSIGNMENT 2

COMP 202, Winter 2020

Due: Wednesday, February 26^{th} 2020, (23:59)

Please read the entire PDF before starting. You must do this assignment individually.

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure.

To get full marks, you must:

- Follow all directions below
  - In particular, make sure that all file names, function names, and variable names are **spelled exactly** as described in this document. Otherwise, a 50% penalty will be applied.
- Make sure that your code runs.
  - Code with errors will receive a very low mark.
- Write your name and student ID as a comment in all .py files you hand in
- Name your variables appropriately
  - The purpose of each variable should be obvious from the name
- Comment your work
  - A comment every line is not needed, but there should be enough comments to fully understand your program
- Avoid writing repetitive code, but rather call helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Write a function `swap` which takes as input two int values `x` and `y`. Your function should do 3 things:

1. Print the value of `x` and `y`
2. Swap the values of the variables `x` and `y`, so that whatever was in `x` is now in `y` and whatever was in `y` is now in `x`
3. Print the value of `x` and `y` again.

For example, if your function is called as follows: `swap(3,4)` the effect of calling your method should be the following printing

```
inside swap:  x is:3 y is:4
inside swap:  x is:4 y is:3
```

Warm-up Question 2 (0 points)

Consider the program you have just written. Create two global integer variables in the main body of your program. Call them `x` and `y`. Assign values to them and call the swap function you wrote in the previous part using `x` and `y` as input parameters.

After calling the `swap()` function —inside the main body— print the values of `x` and `y`. Are they different than before? Why or why not?

Warm-up Question 3 (0 points)

Create a function called `counting` that takes as input a positive integer and counts up to that number. For example:

```
>>> counting(10)
Counting up to 10: 1 2 3 4 5 6 7 8 9 10
```

Warm-up Question 4 (0 points)

Modify the last function by adding an additional input that reprents the step size by which the function should be counting. For example:

```
>>> counting(25, 3)
Counting up to 25 with a step size of 3: 1 4 7 10 13 16 19 22 25
```

Warm-up Question 5 (0 points)

Write a function `replace_all` which takes as input a string and two characters. If the second and third input string do not contain exactly one character the function should raise a `ValueError`. Otherwise, the function returns the string composed by the same characters of the given string where all occurrences of the first given character are replaced by the second given character. For example, `replace_all("squirrel", "r" , "s")` returns the string `"squissel"`, while `replace_all("squirrel", "t", "a")` returns the string `"squirrel"`. Do not use the method `replace` to do this.

Warm-up Question 6 (0 points)

Write a module with the following global variables:

```
lower_alpha = "abcdefghijklmnopqrstuvwxyz"
upper_alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

In this module write a function `make_lower` which takes a string as input and returns a string containing the same characters as the input string, but all in lower case. For example, `make_lower("AppLE")` returns the string `"apple"`. Do not use the method `lower` to do this. Hint: note that characters from the English alphabet appear in the same position in the two global variables.

Warm-up Question 7 (0 points)

Create a function called `generate_random_list` which takes an input an integer `n` and returns a list containing `n` random integers between 0 and 100 (both included). Use `random` to do this.

Warm-up Question 8 (0 points)

Write a function `sum_numbers` which takes as input a list of integers and returns the sum of the numbers in the list. Do not use the built-in function `sum` to do this.

Part 2

The questions in this part of the assignment will be graded.
The main learning objectives for this assignment are:

- Correctly define and use simple functions.
- Solidify your understanding of the difference between `return` and `print`.
- Generate and use random numbers inside a program.
- Understand how to test functions that contain randomness
- Correctly use loops and understand how to choose between a while and a for loop.
- Solidify your understanding of how to work with strings: how to check for membership, how to access characters, how to build a string with accumulator patterns.
- Begin to use very simple lists.
- Create a program with more than one module.
- Learn how to use functions you have created in a different module.

Note that the assignment is designed for you to be practicing what you have learned in class up to Thursday Feb 13th. For this reason, **you are NOT allowed** to use what we have not seen in class (for example, you cannot use dictionaries, or the built-in functions `ord` and `chr`). You will be heavily penalized if you do so.

For full marks on both questions, make sure to add the appropriate documentation string (docstring) to *all* the functions you write. The docstring must contain the following:

- The type contract of the function.
- A brief description of what the function is expected to do.
- At least 3 examples of calls to the function. Make sure to include examples for the different scenarios including edge cases if needed. You are allowed to use *at most* one example per function from this pdf.

Please note that all the examples are given as if you were to call the functions from the shell.

Question 1: Craps (40 points)
Craps is a dice game where each player bets on the outcome of the dice rolls. The goal of this question is to write several functions in order to create a program that simulates the outcome of a Pass Line Bet (see below) in a game of Craps. All the code for this question must be placed in a file named `dice_game.py`.

The Pass Line Bet

Craps is a game of rounds. The first dice roll of a round is called the Come-out Roll. When a player is making a Pass Line Bet, they will place a bet before the Come-Out Roll. Depending on the result of the roll, the player might win, lose, or go to the "next stage" of the game:

- If a 7 or an 11 is rolled, then the player wins.
- If a 2, 3, or 12 is rolled, then the player loses.
- If any other number is rolled, the player must go to the next stage.

For the next stage, it is important to remember which number was rolled in the Come-Out Roll, this number is called the *point.* In the second stage, the player *will keep rolling the dice* unless one of the following happens:

- A 7 is rolled, and the player loses the bet
- The *point* is rolled again, and the player wins the bet

The payout is 1:1: the players win as much as they bet. Thus, if the player bets $5 and wins they receive an additional $5. If they lose, they lose the entire bet.

Let's see a couple of examples:

- The result of the Come-Out Roll is a 3. → The player loses!
- The result of the Come-Out Roll is a 5 → The dice are rolled again until either a 7 or a 5 is rolled. Supposed that the results of the rolls are the following: 10, 11, 4, 7. → The player loses!
- The result of the Come-Out Roll is a 9 → The dice are rolled again until either a 7 or a 9 is rolled. Let the results of the rolls be as follow: 3, 5, 9. → The player wins!

Now that we now how the game works, let's see which functions we need to simulate the result of a Pass Line bet in a game of Craps. Because this program simulates a game of dice, you will need to generate random numbers. To achieve this, make sure to import `random` at the top of your file.

For full marks, all the following functions must be part of your program:

- `dice_roll`: In a game of Craps, players are betting on the outcome of a roll of two six-sided dice. Write a function called `dice_roll` that simulates the roll of two six-sided dice. Such function takes no input and returns an integer between 2 and 12 (included), which is the sum of the result of rolling two six-sided dice. Notice, that to simulate the roll of two six-sided dice, you will have to generate two random numbers between 1 and 6 (both included) and sum the results together. If you have any doubts on how to achieve this, review the slide from lecture 7. To be able to test the correctness of your function, we suggest you to fix a seed while testing the function. **Do NOT use `random.seed` from within the function `dice_roll`.** Once you have created the function, if you test if from the shell you should see the following:

```
>>> random.seed(5)
>>> dice_roll()
8
>>> dice_roll()
9
>>> random.seed(2)
>>> dice_roll()
2
>>> dice_roll()
4
```

- `second_stage`: This function simulates the second stage of the Pass Line Bet. It takes as input one integer value that corresponds to the *point*, the number rolled in the Come-Out Roll (either 4, 5, 6, 8, 9, or 10), and *returns* an integer which will either be a 7 or the point itself depending on which one gets rolled first. The function should also *print* (on the same line!) the result of all the dice rolls carried out before either a 7 or the point is rolled. For example,

```
>>> random.seed(5)
>>> r = second_stage(6)
8 9 12 11 5 8 3 4 6
>>> r
6
>>> random.seed(789)
>>> r = second_stage(8)
10 7
>>> r
7
```

  Make sure to use `dice_roll` to obtain the value of each roll.

- `can_play`: This function takes two floats as input and returns a boolean value. The first input value corresponds to the money the player has, the second corresponds to how much money the player would like to bet. A player is allowed to play **only if** they bet more than $ 0.0, but not more than what they own. If the player is allowed to play, the function *returns* `True`, otherwise it returns `False`. For example,

  ```
  >>> can_play(5.25, 5.0)
  True
  >>> can_play(0.0, 2.0)
  False
  >>> can_play(5.0, -3.0)
  False
  ```

- `pass_line_bet`: This function simulates what happens when a Pass Line Bet is placed. It takes two floats as input: the first one corresponds the total amount of money the player has, the second correspond to how much money the player decides to bet. You can assume that the given values are such that the player can play (see previous function). The function returns a float which corresponds to the amount of money the player has left after one round of Craps.

  The function should *display* the result of the Come-Out Roll (the first roll in a round of Craps) as well as what will happen next. Recall that the player wins with a 7 or 11, loses with a 2, 3, or 12, and moves to the second stage with any other number. Here are three possible statements that the function might display after the Come-Out Roll:

  ```
  A 7 has been rolled. You win!
  A 12 has been rolled. You lose!
  A 5 has been rolled. Roll again!
  ```

  If necessary, the function should simulate the second stage in order to determine whether the player wins or loses. If at the end of the second stage a 7 was rolled the function *displays* a statement informing the player they lost, if instead the point was rolled the function *displays* a statement informing the player they won.

  For full marks, you should use `second_stage`, and `dice_roll` in order to implement the simulation of a Pass Line Bet. Remember to return the amount of money the player has left depending on whether they won or lost the bet. For example,

  ```
  >>> random.seed(5)
  >>> m = pass_line_bet(12.5, 3.5)
  A 8 has been rolled. Roll again!
  9 12 11 5 8
  You win!
  >>> m
  16.0
  >>> random.seed(789)
  >>> m = pass_line_bet(12.5, 3.5)
  A 10 has been rolled. Roll again!
  7
  You lose
  >>> m
  9.0
  >>> random.seed(3)
  >>> m = pass_line_bet(5.0, 5.0)
  A 7 has been rolled. You win!
  >>> m
  10.0
  ```

- `play`: This function takes no input and returns no value. The function retrieves two *inputs from the user*. The first input corresponds to the money the player has, the second to the money they would like to bet. If the user does not have enough money to play, the function displays a message informing the user about it and terminates. Otherwise, the function calls `pass_line_bet` with the appropriate inputs in order to place the bet. At the end, make sure to *display* a statement informing the player about how much money they have left after their bet. The number representing the money left should not have more than 2 decimals.

  For example,

```
>>> play()
Please enter your money here: 3.0
How much would you like to bet? 5.0
Insufficient funds. You cannot play.

>>> random.seed(5)
>>> play()
Please enter your money here: 12.5
How much would you like to bet? 3.5
A 8 has been rolled. Roll again!
9 12 11 5 8
You win!
You now have $16.0

>>> random.seed(789)
>>> play()
Please enter your money here: 12.5
How much would you like to bet? 3.5
A 10 has been rolled. Roll again!
7
You lose
You now have $9.0

>>> random.seed(3)
>>> play()
Please enter your money here: 12.5
How much would you like to bet? 3.5
A 7 has been rolled. You win!
You now have $16.0
```

Question 2: Ciphers (60 points)

Caesar's cipher is a very well known and simple encryption scheme. The point of an encryption scheme is to transform a message so that only those authorized will be able to read it. Caesar's cipher conceal a message by replacing each letter in the original message, by a letter corresponding to a certain number of letters to the right on the alphabet. Of course the message can be retrieved by replacing each letter in the encoded message (the ciphertext) with the letter corresponding to the same number of position to the left on the alphabet. To achieve this, the cipher has a key that needs to be kept private. Only those with the key can encode and decode a message. Such a key determines the shift that needs to be performed on each letter. For example, here is how a string containing the entire alphabet will be encrypted using a key equal to 3:

Original:	abcdefghijklmnopqrstuvwxyz
Encrypted:	defghijklmnopqrstuvwxyzabc

Vigenère's cipher is a slightly more complex encryption scheme, also used to transform a message. The key of this cipher consists of a word and the cipher works by applying different shifts to the letters in the message based on the letter of the keyword. Each letter can be associated with a number corresponding to its position in the English alphabet (starting to count from 0). For instance, the letter 'a' is associated to 0, 'c' to 2, and 'z' to 25. Therefore, the keyword of the cipher will provide as many integers as letters in the word and these integers will be used to implement different shifts. Let's see how: suppose the message to encrypt is "elephants" and the keyword is "rats". The first thing to do is to repeat the keyword until its length matches the one of the message.

Message:	e	l	e	p	h	a	n	t	s
Keyword:	r	a	t	s	r	a	t	s	r

Now, each letter of "ratsratsr" is associated to both a letter in the message and an integer. We can encrypt each letter of the message using a shift that corresponds to the integer associated to it through the keyword. In this case 'r' corresponds to 17, so the first letter of the message which is an 'e' will be encrypted using a 'v', the second letter 'l' as an 'l' since 'a' is associated to 0, and so on. The entire message will be encrypted as "vlxhyaglj".

The goal of this exercise is to write two modules with several functions in order to create a program that encodes and decodes messages using Caesar's and Vigenère's ciphers.

The helper functions

Let's start by creating a module called `crypto_helpers` which contains several helper functions that we need to use to implement the ciphers. At the beginning of the file, create a global variable as follows:

```
ALPHABET = ‘abcdefghijklmnopqrstuvwxyz’
```

Note that we used all caps for the name of the variable because this is meant to represent an constant in your program.

For full marks, all the following functions must be part of this module:

- `in_engl_alpha`: This function takes a string as input and returns true if this is a non-empty string containing only characters from the English alphabet, false otherwise. Note that this function should not be case sensitive. For example,

```
>>> in_engl_alpha(‘a’)
True
>>> in_engl_alpha(‘G’)
True
>>> in_engl_alpha(‘cat’)
True
>>> in_engl_alpha(‘cats and dogs’)
False
```

```
>>> in_engl_alpha('@')
False
>>> in_engl_alpha('è')
False
```

- `shift_char`: This function takes a string representing a single character as input, and an integer `n`. The function should verify that the string received represents a single character, if not a `ValueError` with the appropriate error message should be raised. Otherwise, if the character is a letter of the English alphabet, the function returns the *lower case* letter which appears `n` position later in the alphabet. If the character received as input is not a letter of the English alphabet, the function returns the character itself with no modification. For example,

```
>>> shift_char('a', 3)
'd'
>>> shift_char('z', 2)
'b'
>>> shift_char('A', -2)
'y'
>>> shift_char('@', 12)
'@'
>>> shift_char('g', 86)
'o'
>>> shift_char('cat', 5)
Traceback (most recent call last):
ValueError: the input string should contain a single character
```

- `get_keys`: This function takes a string as input and returns an list of integers. The elements of the list correspond to the position (counting from 0) of each character in the string as a letter of the English alphabet. If the string received as input is a non-empty string containing characters other than letters from the English alphabet, then the function should raise a `ValueError` with the appropriate error message. For example,

```
>>> get_keys('hello')
[7, 4, 11, 11, 14]
>>> get_keys('AbC')
[0, 1, 2]
>>> get_keys('')
[]
>>> get_keys('c@t')
Traceback (most recent call last):
ValueError: the input string must contain only characters from the English alphabet.
```

- `pad_keyword`: This function takes as input a string and an integer `n`. It returns a string of length `n` obtained by concatenating characters of the input string together until the desire length is matched. Note that `n` can be smaller than the length of the input string. If the input string is empty, the function raise a `ValueError` with the appropriate message. For example,

```
>>> pad_keyword('cat', 10)
'catcatcatc'
>>> pad_keyword('artichoke', 5)
'artic'
```

The ciphers

Let's now create a module called `ciphers`. In this module you will write the functions that implement the Caesar's and the Vigenère's cipher. To do so, make sure to import the `crypto_helpers` module so that you can use the helper functions listed above.

For full marks, all the following functions must be part of this module:

- `caesar`: This function takes as input a string (the message to encrypt/decrypt), a integer `k` (the key of the cipher), and another integer `m` representing the mode (encrypt/decrypt). If `m` is 1 the function will be encrypting the message, if instead it is -1 the function will be decrypting the message. If it has any other value, the function raises a `ValueError` indicating that no other mode is supported. This function returns a string obtained by encrypting or decrypting (depending on `m`) the message received as input using the Caesar's cipher with key `k`. For example,

```
>>> caesar('abc', 10, 1)
'klm'
>>> caesar('wtaad', 15, -1)
'hello'
>>> caesar('apple', -2, 1)
'ynnjc'
>>> caesar('cats and dogs', 5, 1)
'hfyx fsi itlx'
>>> caesar('hello', 11, 5)
Traceback (most recent call last):
ValueError: mode not supported
```

- `vigenere`: This function takes as input a string representing the message to encrypt/decrypt, another string representing the key of the cipher, and an integer `m` representing the mode (encrypt or decrypt). If `m` is 1 the function will be encrypting the message, if instead it is -1 the function will be decrypting the message. If it has any other value, the function raises a `ValueError` indicating that no other mode is supported. This function returns a string obtained by encrypting or decrypting (depending on `m`) the message received as input using the Vigenère's cipher with key received as input. Go back at the beginning of this section to review how Vigenère's cipher works. Note that this function will raise an error if the string representing the key is empty. For example,

```
>>> vigenere('BaNAna', 'apple', 1)
'bpclra'
>>> vigenere('bpclra', 'apple', -1)
'banana'
>>> vigenere('elephants and hippos', 'rats', 1)
'vlxhyaglj tfu aagphk'
>>> vigenere('vlxhyaglj tfu aagphk', 'RATS', -1)
'elephants and hippos'
>>> vigenere('hello', 'cat', 5)
Traceback (most recent call last):
ValueError: mode not supported
```

What To Submit

You must submit all your files on codePost (https://codepost.io/). The file you should submit are listed below. Any deviation from these requirements may lead to lost marks.

`dice_game.py`
`crypto_helpers.py`
`ciphers.py`
`README.txt` In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your program, it may lead the TA to give you more partial credit.
This file is also where you should make note of anybody you talked to about the assignment. Remember this is an individual assignment, but you can talk to other students using the **Gilligan's Island Rule**: you can't take any notes/writing/code out of the discussion, and afterwards you must do something inane like watch television for at least 30 minutes.
If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.